Data Manipulation with R

Journalism 303: An introduction to the dplyr package

Martin Frigaard

2021-09-21

Objectives

1) Common data manipulation tasks

2) `dplyr`'s verbs

3) the pipe `%>%`

Materials

Follow along with the exercises:

https://mjfrigaard.github.io/csuc-data-journalism/lessons-exercises.html

A web version of these slides is located:

https://mjfrigaard.github.io/csuc-data-journalism/slides.html

What are common data manipulation tasks?

1. Viewing the dataset

2. Choosing columns/rows

3. Ordering rows

4. Changing existing columns

5. Creating or calculating new columns

dplyr = a grammar for data manipulation

`dplyr` = "dee + ply + ARRRR"

Pliers **are tools for grasping or manipulating common objects**

The `dplyr` package has a variety of verbs for performing common data manipulations

The `starwars` dataset

These data come from the Star Wars API:

Read more about the data here:

https://dplyr.tidyverse.org/reference/starwars.html

Load the `starwars` dataset

The `starwars` data comes from the `dplyr` package, so we can access it using the code below:

```
install.packages("dplyr")
library(dplyr)
dplyr::starwars
```

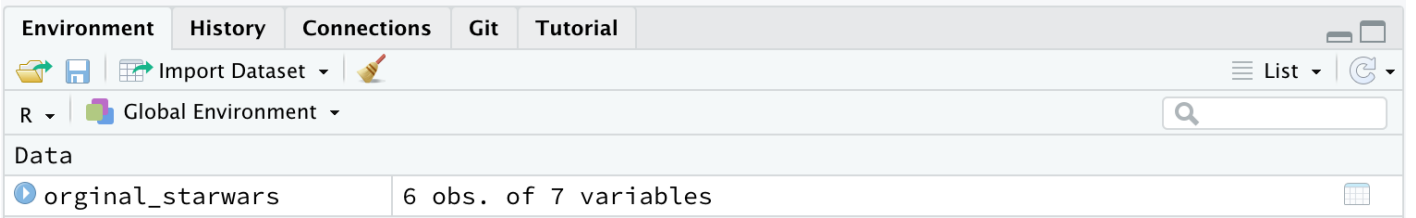
We'll use a smaller version of this dataset (`original_starwars`) to show `dplyr`'s common data manipluation verbs

Import `original_starwars`

Import the data using the url and `readr`

```
library(readr)
original_starwars <- read_csv("https://bit.ly/mini-strwrs")
```

This loads the dataset into our *Environment* pane



`dplyr` verbs

The primary verbs for data manipulation in `dplyr`:

`glimpse()`

`select()`

`filter()`

`arrange()`

`mutate()`

Viewing the data = `glimpse()`

We need to view the data we're manipulating to see if the changes are correct

View = `glimpse()`

Take a look at the entire dataset using `dplyr::glimpse()`

```
glimpse(original_starwars)
```

```
## Rows: 6
## Columns: 6
## $ name       <chr> "Luke Skywalker", "C-3PO", "R2-D2", "Le…
## $ height     <dbl> 172, 167, 96, 150, 228, 180
## $ mass       <dbl> 77, 75, 32, 49, 112, 80
## $ hair_color <chr> "blond", NA, NA, "brown", "brown", "bro…
## $ species    <chr> "Human", "Droid", "Droid", "Human", "Wo…
## $ homeworld  <chr> "Tatooine", "Tatooine", "Naboo", "Alder…
```

`glimpse()` transposes the data and prints as much of it to the screen as possible

View the data in the *Console*

Enter the name of the dataset to print it to the Console

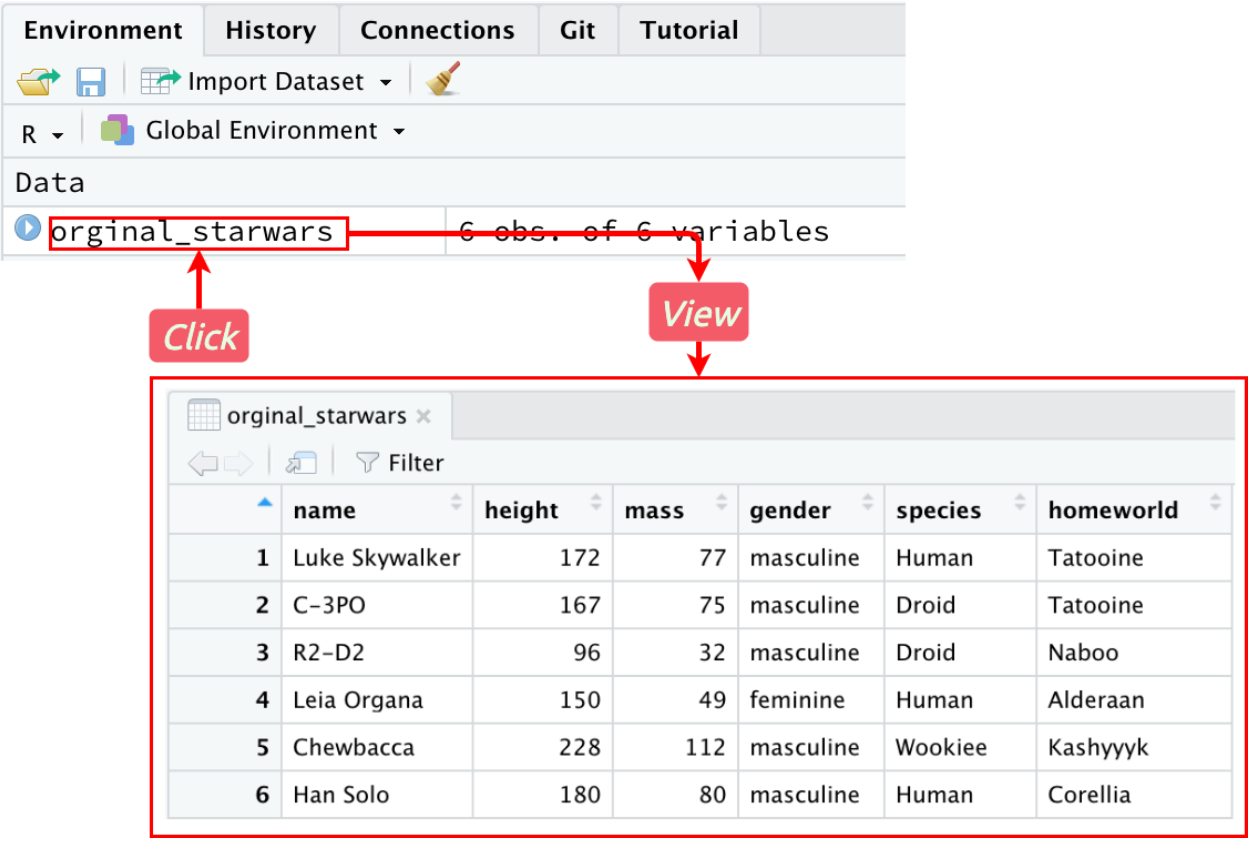
```
original_starwars
```

name	height	mass	hair_color	species	homeworld
<chr>	<dbl>	<dbl>	<chr>	<chr>	<chr>
Luke Skywalker	172	77	blond	Human	Tatooine
C-3PO	167	75	*NA*	Droid	Tatooine
R2-D2	96	32	*NA*	Droid	Naboo
Leia Organa	150	49	brown	Human	Alderaan
Chewbacca	228	112	brown	Wookiee	Kashyyyk
Han Solo	180	80	brown	Human	Corellia

6 rows

View the data in the *Data Viewer*

View the `original_starwars` dataset using RStudio's data editor



	name	height	mass	gender	species	homeworld
1	Luke Skywalker	172	77	masculine	Human	Tatooine
2	C-3PO	167	75	masculine	Droid	Tatooine
3	R2-D2	96	32	masculine	Droid	Naboo
4	Leia Organa	150	49	feminine	Human	Alderaan
5	Chewbacca	228	112	masculine	Wookiee	Kashyyyk
6	Han Solo	180	80	masculine	Human	Corellia

Choosing columns = `select()`

Choose columns with `select()`

`select()` allows us to pick specific columns out of a dataset

```
select(original_starwars, name, homeworld, species)
```

name	homeworld	species
<chr>	<chr>	<chr>
Luke Skywalker	Tatooine	Human
C-3PO	Tatooine	Droid
R2-D2	Naboo	Droid
Leia Organa	Alderaan	Human
Chewbacca	Kashyyyk	Wookiee
Han Solo	Corellia	Human

6 rows

Choose columns with `select()`

We can use negation (`-`) to remove columns

```
select(original_starwars, -c(mass, height, hair_color))
```

name	species	homeworld
<chr>	<chr>	<chr>
Luke Skywalker	Human	Tatooine
C-3PO	Droid	Tatooine
R2-D2	Droid	Naboo
Leia Organa	Human	Alderaan
Chewbacca	Wookiee	Kashyyyk
Han Solo	Human	Corellia

6 rows

`select()` helpers

`select()` comes with 'helpers' to make chosing columns easier (and reduces typing!)

Helper	Outputs
`starts_with()`	choose columns starting with...
`ends_with()`	choose columns ending with...
`contains`	choose columns with names containing...
`matches()`	choose columns matching regex...
`one_of()`	choose columns from a set of names...
`num_range()`	choose columns from a numerical index...

Choose columns with `select()`

Select columns using `matches()`

```
select(original_starwars, name, matches("_"))
```

name <chr>	hair_color <chr>
Luke Skywalker	blond
C-3PO	*NA*
R2-D2	*NA*
Leia Organa	brown
Chewbacca	brown
Han Solo	brown

6 rows

See the `select()` exercises for more examples!

Choosing rows with `filter()`

Choose rows with `filter()`

`filter()` lets us pull out rows based on logical conditions

```
filter(original_starwars, species == "Human")
```

name	height	mass	hair_color	species	homeworld
<chr>	<dbl>	<dbl>	<chr>	<chr>	<chr>
Luke Skywalker	172	77	blond	Human	Tatooine
Leia Organa	150	49	brown	Human	Alderaan
Han Solo	180	80	brown	Human	Corellia

3 rows

Choose rows with `filter()`

`filter()` logical conditions include:

Logical Test	Outputs
`<`	Less than
`>`	Greater than
`==`	Equal to
`<=`	Less than or equal to
`>=`	Greater than or equal to
`!=`	Not equal to
`%in%`	Group membership
`is.na()`	is NA (missing)
`!is.na()`	is not NA (non-missing)

Choose rows with `filter()`

Combine logical condtions with `&` or `,`

this gets the same results...

```
filter(original_starwars,
          species == "Human" & !is.na(hair_color))
```

name	height	mass	hair_color	species	homeworld
<chr>	<dbl>	<dbl>	<chr>	<chr>	<chr>
Luke Skywalker	172	77	blond	Human	Tatooine
Leia Organa	150	49	brown	Human	Alderaan
Han Solo	180	80	brown	Human	Corellia

3 rows

Choose rows with `filter()`

Combine logical condtions with & or ,

...as this

```
filter(original_starwars,
          species == "Human" , !is.na(hair_color))
```

name	height	mass	hair_color	species	homeworld
<chr>	<dbl>	<dbl>	<chr>	<chr>	<chr>
Luke Skywalker	172	77	blond	Human	Tatooine
Leia Organa	150	49	brown	Human	Alderaan
Han Solo	180	80	brown	Human	Corellia

3 rows

Choose rows with `filter()`

Remember that *any* logical condition works for `filter()`ing, so we can borrow functions from other packages to help us

```
filter(original_starwars,
       str_detect(string = name, pattern = "[:digit:]"))
```

The `stringr::str_detect()` function returns a logical condition, so we can use it *inside* `filter()`

name	height	mass	hair_color	species	homeworld
<chr>	<dbl>	<dbl>	<chr>	<chr>	<chr>
C-3PO	167	75	*NA*	Droid	Tatooine
R2-D2	96	32	*NA*	Droid	Naboo

2 rows

See the `filter()` exercises for more examples!

Sorting rows with `arrange()`

Sort rows with `arrange()`

`arrange()` sorts the contents of a dataset (ascending or descending)

```
arrange(original_starwars, height)
```

name	height	mass	hair_color	species	homeworld
<chr>	<dbl>	<dbl>	<chr>	<chr>	<chr>
R2-D2	96	32	*NA*	Droid	Naboo
Leia Organa	150	49	brown	Human	Alderaan
C-3PO	167	75	*NA*	Droid	Tatooine
Luke Skywalker	172	77	blond	Human	Tatooine
Han Solo	180	80	brown	Human	Corellia
Chewbacca	228	112	brown	Wookiee	Kashyyyk

6 rows

Sort rows with `arrange()`

`arrange()`'s default is to sort ascending--include `desc()` to sort descending

```
arrange(original_starwars, desc(height))
```

name	height	mass	hair_color	species	homeworld
<chr>	<dbl>	<dbl>	<chr>	<chr>	<chr>
Chewbacca	228	112	brown	Wookiee	Kashyyyk
Han Solo	180	80	brown	Human	Corellia
Luke Skywalker	172	77	blond	Human	Tatooine
C-3PO	167	75	*NA*	Droid	Tatooine
Leia Organa	150	49	brown	Human	Alderaan
R2-D2	96	32	*NA*	Droid	Naboo

6 rows

See the `arrange()` exercises for more examples!

Create columns with `mutate()`

`mutate()` allows us to create new columns


```
mutate(original_starwars,
            # create new bmi variable
            bmi = mass / ((height / 100) ^ 2))
```

name	height	m...	hair_color	species	homewo...	bmi
<chr>	<dbl>	<dbl>	<chr>	<chr>	<chr>	<dbl>
Luke Skywalker	172	77	blond	Human	Tatooine	26.02758
C-3PO	167	75	*NA*	Droid	Tatooine	26.89232
R2-D2	96	32	*NA*	Droid	Naboo	34.72222
Leia Organa	150	49	brown	Human	Alderaan	21.77778
Chewbacca	228	112	brown	Wookiee	Kashyyyk	21.54509
Han Solo	180	80	brown	Human	Corellia	24.69136

6 rows

Create columns with `mutate()`

`mutate()` allows us to change existing columns, too

```
mutate(original_starwars,
             # create bmi
             bmi = mass / ((height / 100) ^ 2),
             # change bmi
             bmi = round(bmi, digits = 0))
```

name	height	mass	hair_color	species	homeworld	bmi
<chr>	<dbl>	<dbl>	<chr>	<chr>	<chr>	<dbl>
Luke Skywalker	172	77	blond	Human	Tatooine	26
C-3PO	167	75	*NA*	Droid	Tatooine	27
R2-D2	96	32	*NA*	Droid	Naboo	35
Leia Organa	150	49	brown	Human	Alderaan	22
Chewbacca	228	112	brown	Wookiee	Kashyyyk	22
Han Solo	180	80	brown	Human	Corellia	25

6 rows

See the `mutate()` exercises for more examples!

Write clearer code with the pipe

`%>%`

The pipe (%>%)

The pipe comes from the `magrittr` package:

https://magrittr.tidyverse.org/

The pipe makes our code easier to read (and write)

Create pipes easily with keyboard shortcuts

Mac

`Cmd` + `Shift` + `M`

Windows

`Crtl` + `Shift` + `M`

How the pipe (%>%) works

Without the pipe, we have to constantly assign the output to new object:

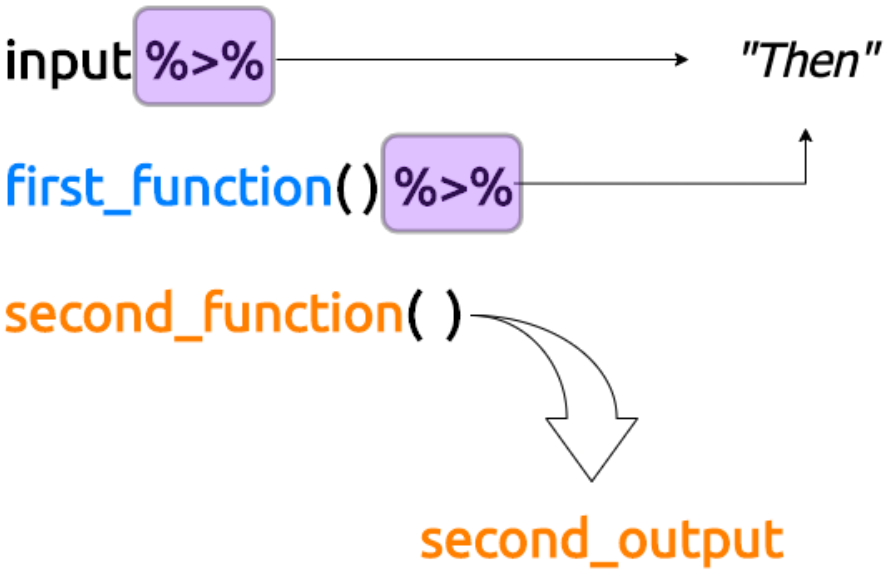
```
first_output <- first_function( input )

second_output <- second_function( first_output )
```

Or use nested functions:

```
second_output  <- second_function( first_function( input ) ,  first_output )
```

How the pipe (%>%) works

The pipe allows us to pass the output from functions left-to-right



%>% can be read as "then"

Creating pipelines of functions

Review the code below and think about what each object contains:

1. Filter `original_starwars` to only brown-haired characters over 100 cm tall
2. Create a `bmi` column using: `mass / ((height / 100) ^ 2)`
3. Select `name`, `bmi`, and `homeworld`
4. Arrange the data by `bmi`, descending

```
object_01 <- filter(original_starwars,
                    hair_color == "brown" & height > 100)
object_02 <- mutate(object_01, bmi = mass / ((height / 100) ^ 2))
object_03 <- select(object_02, name, bmi, homeworld)
object_04 <- arrange(object_03, desc(bmi))
```

Creating pipelines of functions

Re-write these functions into a pipeline, ending with a single output (new_original_starwars)

1. Filter `original_starwars` to only brown-haired characters over 100 cm tall
2. Create a `bmi` column using: `mass / ((height / 100) ^ 2)`
3. Select `name`, `bmi`, and `homeworld`
4. Arrange the data by `bmi`, descending

```
original_starwars %>%
  filter(hair_color == "_____" & height > ___) %>%
  mutate(___ = mass / ((height / 100) ^ 2)) %>%
  select(____, bmi, _________) %>%
  arrange(____(bmi)) -> new_original_starwars
```

Creating pipelines of functions

The answer is below:

```
original_starwars %>%
  filter(hair_color == "brown" & height > 100) %>%
  mutate(bmi = mass / ((height / 100) ^ 2)) %>%
  select(name, bmi, homeworld) %>%
  arrange(desc(bmi)) -> new_original_starwars
new_original_starwars
```

name	bmi	homeworld
<chr>	<dbl>	<chr>
Han Solo	24.69136	Corellia
Leia Organa	21.77778	Alderaan
Chewbacca	21.54509	Kashyyyk

3 rows

See the `pipe` exercises for more examples!

Resources for Data Manipluation

1. R for Data Science

2. Data Wrangling with R

3. Stack Overflow questions tagged with **`dplyr`**

4. RStudio Community posts tagged **`dplyr`**